第二章 进程管理

习题

2.8 在生产者—消费者问题中，如果缺少了 signal(full)或 signal(empty)，对执行结果会有何影响？

参考答案：（1）在生产者—消费者问题中，如果缺少了 signal(full)，则消费者一开始便会因 wait(full)自我阻塞和无法向前推进，而伴随生产者生产数据、执行 wait(empty)并放到循环缓冲的空缓冲区，直到循环缓冲的所有缓冲区都变成满缓冲区（期间 full 信号量却始终保持为 0）之后，empty 信号量也演化为 0，生产者进程同样因 wait(empty) 自我阻塞和无法向前推进，生产者进程和消费者进程陷入死锁。（2）如果缺少了 signal(empty)，则执行起初，伴随生产者生产数据、执行 wait(empty)和放到循环缓冲的空缓冲区并通过 signal(full)通知和唤醒消费者进程从满缓冲区中提取数据，但消费者进程并不执行 signal(empty)，故而当生产者生产和放入 n 个缓冲区的数据后，empty 信号量演化为 0，生产者进程将因再度执行 wait(empty) 自我阻塞和无法向前推进，同时当消费者进程取完 n 个缓冲区中的数据后，其 full 信号量也会演化为 0，再度执行同样因 wait(full)自我阻塞和无法向前推进，生产者进程和消费者进程陷入死锁。

2.9 在生产者—消费者问题中，如果将两个 wait 操作即 wait(full)和 wait(mutex)互换位置；或者是将 signal(mutex)与 signal(full)互换位置，结果会如何？

参考答案：（1）在生产者—消费者问题中，如果将两个 wait 操作即 wait(full)和 wait(mutex)互换位置，则若一开始（或者执行中循环缓冲缓冲区全为空缓冲区的情况下）由消费者进程率先获得执行权，通过 wait(mutex)获得循环缓冲临界资源访问权后，必然在执行下一句即 wait(full)时自我阻塞和无法向前推进，而生产者进程通过 wait(empty)申请和获得空缓冲区资源后，也会因 wait(mutex)无法获得循环缓冲临界资源访问权、自我阻塞和无法向前推进，从而使二者陷入死锁。（2）若是将 signal(mutex)和 signal(full)互换位置，只是临界资源访问权与满缓冲区资源信号量释放次序的变更，不会引发死锁等后果。

2.10 我们为某临界资源设置一把锁 W，当 W=1 时，表示关锁；W=0 时，表示锁已打开。试写出开锁原语与关锁原语，并利用它们去实现互斥。

参考答案：

```
VAR W : int := 0;
1、关锁原语 Lock(W)：
    while W=1 do no_op;
    W:=1;
2、开锁原语 Unlock(W)：
    W:=0;
3、互斥实现：
    repeat
```

……
Lock(W);
临界区
Unlock(W);
……
until false

2.11 试修改下面生产者—消费者问题解法中的错误：

```
producer:
 begin
   repeat
      ……
      produce an item in nextp;
      wait(mutex);
      wait(full);
      buffer(in):=nextp;
      signal(mutex);
   until false;
 end

consumer:
 begin
   repeat
      wait(mutex);
      wait(empty);
      nextc:=buffer(out);
      out:=out+1;
      signal(mutex);
      consume item in nextc;
   until false;
 end
```

参考答案：

```
Var   buffer: array [0..n-1] of item;
      in, out: integer := 0, 0;
      mutex, empty, full : semphore := 1, n, 0 ;
begin
   parbegin
      producer1; …; produceri; …; producerY;
      consumer1; …; consumerj; …; consumerX;
   parend
end
```

$producer_i$:

```
Var nextp: item;
   begin
```

repeat
produce an item in nextp;
~~wait(mutex);~~ **wait(empty);**
~~wait(full);~~ **wait(mutex);**
~~buffer(in):=nextp;~~ **buffer[in] := nextp; in := (in+1) mod n;**
signal(mutex);
signal(full);
until false;
end
consumer$_j$:
Var nextc: item;
begin
repeat
~~wait(mutex);~~ **wait(full);**
~~wait(empty);~~ **wait(mutex);**
~~nextc:=buffer(out);~~ **nextc:=buffer[out];**
~~out:=out+1;~~ **out := (out+1) mod n;**
signal(mutex);
signal(empty);
consume the item in nextc;
until false;
end

2.12 试利用记录型信号量写出一个不会出现死锁的哲学家进餐问题的算法。

参考答案 1（奇数号哲学家和偶数号哲学家拿左右两边筷子的先后次序恰好相反）：

Var chopstick: array[0..4] of semphore:=(1,1,1,1,1);
begin
parbegin
philosophy$_0$;
philosophy$_1$;
philosophy$_2$;
philosophy$_3$;
philosophy$_4$;
parend
end
philosophy$_i$:（i=0, 2, 4）
begin
repeat
Think;
wait(chopstick[(i+1)mod 5]);
wait(chopstick[i]);
Eat;
signal(chopstick[i]);
signal(chopstick[(i+1)mod 5]);

```
        until false;
    end
philosophyj :（i=1, 3）
    begin
        repeat
            Think;
            wait(chopstick[i]);
            wait(chopstick[(i+1)mod 5]);
            Eat;
            signal(chopstick[(i+1)mod 5]);
            signal(chopstick[i]);
        until false;
    end
```

参考答案 2（限定最多 4 个哲学家同时拿起筷子吃饭）：

```
Var chopstick: array[0..4] of semphore:=(1,1,1,1,1); semCount: semaphore:=4;
begin
    parbegin
        philosophy0 ;
        … ;    philosophyi ;    …
        philosophy4;
    parend
end
philosophyi :
    begin
        repeat
            Think;
            wait(semCount);
            wait(chopstick[i]);
            wait(chopstick[(i+1)mod 5]);
            Eat;
            signal(chopstick[(i+1)mod 5]);
            signal(chopstick[i]);
            signal(semCount);
        until false;
    end
```

参考答案 3（通过互斥信号量实现哲学家左右两边筷子的同时拿取）：

```
Var chopstick: array[0..4] of semphore:=(1,1,1,1,1); mutex: semphore :=1;
begin
    parbegin
        philosophy0 ;
        … ;    philosophyi ;    …
        philosophy4;
    parend
end
```

philosophy$_i$:

```
    begin
        repeat
            Think;
            wait(mutex);
            wait(chopstick[i]);
            wait(chopstick[(i+1)mod 5]);
            signal(mutex);
            Eat;
            signal(chopstick[(i+1)mod 5]);
            signal(chopstick[i]);
        until false;
    end
```

2.13 在测量控制系统中的数据采集任务，把所采集数据送一单缓冲区；计算任务从该单缓冲区中取出数据进行计算。写出利用信号量机制实现两者共享单缓冲的同步算法。

标准答案：

```
Var    buffer: item;
       empty, full : semphore := 1, 0;
begin
    parbegin
        data-collecting-task;
        computing-task;
    parend
end
data-collecting-task:
Var data: item;
    begin
        repeat
            采集数据存放在局部变量 data 中;
            wait(empty);
            buffer := data;
            signal(full);
        until false;
    end
computing-task:
Var data: item;
    begin
        repeat
            wait(full);
            data:=buffer;
            signal(empty);
            利用局部变量 data 进行计算;
```

```
        until false;
    end
```

2.14 给出基于记录型信号量机制的写者优先的读者-写者问题的同步解决方案。

参考答案：

```
Var readercount, writercount: integer := 0, 0; //分别为读者计数变量和写者计数变量
        S, mutex,rmutex,wmutex : semphore := 1,1,1,1;
        //S 用于读者与写者的统一排队
        //rmutex 和 mutex 分别用于读者计数变量和写者计数变量的互斥访问
        // wmutex 用于写者与其它写者及读者的互斥执行
begin
    parbegin
        reader1;  … ;  readeri  ;  …  ; readerm;
        writer1;  … ;  writerj  ;  …  ; writern;
    parend
end
readeri:
    begin
        repeat
            wait(S);
            wait(rmutex);
            if readercount=0 then wait(wmutex);
            readercount := readercount +1;
            signal(rmutex);
            signal(S);
            Perform read operation;
            wait(rmutex);
            readercount := readercount -1;
            if readercount=0 then signal(wmutex);
            signal(rmutex);
        until false;
    end
writeri:
    begin
        repeat
            wait(mutex);
            if writercount=0 then wait(S);
            writercount := writercount +1;
            signal(mutex);
            wait(wmutex);
            Perform write operation;
            signal(wmutex);
            wait(mutex);
            writercount := writercount -1;
```

```
            if writercount=0 then signal(S);
            signal(mutex);
        until false;
    end
```

2.15-2.16 点评：大部分同学回答基本正确，但个别同学回答太珍惜笔墨和过于简单化，希改进。有关描述参课件和课本。

2.15 简明扼要地谈谈你对各种进程通信方式的认识与理解，并着重就消息缓冲队列通信机制进行分析与描述。

2.16 为什么要引入管程？并就管程的组成和同步互斥机理展开简明扼要的讨论。